EKS MANAGER · CERTIFICATES

Automated Certificate Renewal on AWS

Let’s Encrypt wildcard certificates, one per DNS zone, issued, stored and renewed inside your own AWS accounts

eksmanager.io | github.com/gitopsmanager/eksmanager-bootstrap

CERTIFICATES · WHAT YOU GET

One certificate per zone, not per cluster

A wildcard for each environment's DNS zone, shared by every cluster in it

Issued centrally

One ACME order per zone from your shared services account, so a growing fleet stays far below Let's Encrypt's 50-per-week limit.

Kept in your account

Full chain and private key in Secrets Manager, one secret per zone. No cert-manager in any cluster.

Served everywhere

Deploy puts it in each namespace that serves the zone. The agent pushes every renewal to them.

```
*.dev.aws.example.com covers one label
yes myapp.dev.aws.example.com
yes client-eks-dev-myapp.dev.aws.example.com
no api.myapp.dev.aws.example.com
```

HOW IT WORKS · DNS

Delegated down to each environment

One name, two zones: public for Let's Encrypt, private for your network

Public · what Let's Encrypt sees

example.com

your public DNS provider

`aws NS ×4 Route 53`

→

aws.example.com

Route 53, public

```
dev  NS ×4 Route 53
prod NS ×4 Route 53
```

→

dev.aws.example.com

Route 53, public

_acme-challenge TXT, only while issuing the wildcard

Private · what your network sees

Active Directory DNS

example.com, internal

```
aws NS ×4 Route 53
forward dev.aws.example.com
  → 10.0.0.10, 10.0.1.10
```

→

Shared private DNS resolver

Route 53 Resolver inbound endpoint

With access to all private hosted zones

→

dev.aws.example.com

Route 53, private

myapp → internal load balancer, written by external-dns

HOW IT WORKS

From a zone list to a served certificate

1 · You push
hosted-zones.json. A sync workflow grants the pipeline each zone's role.

2 · Workflow
lets-encrypt.yml validates the file and uploads a zip to S3 over OIDC.

3 · CodeBuild
Starts on the upload, or every Saturday at 03:00 UTC, and runs Terraform.

4 · Hop
Assumes the zone account's cert_manager role, which you own.

5 · DNS-01
Writes the _acme-challenge TXT in the public zone. Let's Encrypt checks it.

6 · Stored
Chain and key in Secrets Manager, one secret per zone.

7 · Pushed
The agent pushes it to each namespace (roadmap: External Secrets).

8 · Served
Each ingress names dns-zone-certs, and Traefik serves it.

`Secret name: /EKSManagerZones/<dns-zone-prefix>—dns-zone-certs`

HOW IT WORKS · ISSUING

Who does what during issuance

lego runs inside Terraform's ACME provider. Let's Encrypt only answers.

#	Who	Does
1	lego	Makes the private key and the request for *.dev.aws.example.com
2	lego → Let's Encrypt	Opens an order and gets back a DNS-01 challenge
3	lego → Route 53	Writes the _acme-challenge TXT, as the zone's cert_manager role
4	Let's Encrypt	Reads the TXT over public DNS and accepts the challenge
5	lego → Let's Encrypt	Sends the request; Let's Encrypt signs it
6	lego	Downloads the signed certificate and chain, then deletes the TXT
7	Terraform	Writes chain and key to Secrets Manager, keeps a copy in its state
8	Agent (roadmap: External Secrets)	Copies it into each namespace's dns-zone-certs

lego is not a separate program: it is a Go library compiled into the vancluever/acme provider. terraform init downloads that provider from the Terraform Registry, at the version pinned in the lock file, and Terraform runs it as a plugin. The private key is made in CodeBuild and never sent to Let's Encrypt.

HOW IT WORKS · ROLES

Two roles that trust each other

Issuing crosses an account boundary: shared services runs it, your zone account writes DNS

Role	Created by	Assumed by	Does
EKSManagerLetsEncryptRole	setup	CodeBuild only	Runs Terraform, writes the secrets
cert_manager, per zone	you	the role above	Writes the _acme-challenge TXT
EKSManagerLetsEncryptGithubActionsRole	setup	lets-encrypt.yml	Uploads the zip to S3
EKSManagerLetsEncryptPolicySyncRole	setup	sync-crt-mgr-arns.yml	Writes the exact AssumeRole grant

None of these can touch Route 53, EKS or EC2 directly. The pipeline reaches DNS only through a role you control.

HOW IT WORKS · THE ZONE LIST

One file lists every zone

hosted-zones.json

```
{
  "acme-email": "platform-team@example.com",
  "acme-staging": true,
  "hosted-zones": [
    { "dns-zone-prefix": "dev",
      "public-hosted-zone": "dev.aws.example.com",
      "private-hosted-zone": "dev.aws.example.com",
      "account": "123456789012",
      "roles": { "cert_manager":
        "arn:aws:iam::123456789012:role/dev-cert-manager",
        "external_dns": "…" } }
  ]
}
```

dns-zone-prefix names the secret: dev becomes /EKSManagerZones/dev-dns-zone-certs. Unique across the file.

public-hosted-zone is looked up by id, never by name, because a split-horizon private zone shares the name.

cert_manager is your role in the zone's account. The grant to assume it follows the file.

acme-email should be a real, monitored address.

RUNNING IT · FIRST RUN

Start on staging, switch once

acme-staging ships as true, because the two ways to fail are not equal

Staging

Untrusted root: browsers warn and the Argo CD CLI fails. Limits high enough to rebuild freely.

Production

Trusted everywhere. 5 per week for the same hostnames, then a week’s lockout with no override.

1. Run on staging: roles assumable, public zone found, DNS-01 passes, a secret per zone
2. Set acme-staging to false and run again: real certificates replace the staging ones
3. Leave it false: zones added later get real certificates straight away

One flag for every zone. Setting it back to true reissues every certificate from the untrusted root, so step 2 is one-way.

RUNNING IT · RENEWALS

Renewal runs itself, every week

Nothing to run: a schedule applies the last uploaded zip

90 days
certificate life

30 days
left when it reissues

Saturday
03:00 UTC, weekly

~4
tries before expiry

- Most runs change nothing: Terraform reissues only inside the 30-day window
- Every run logs certificate_expiry per zone, the cheapest proof renewal still works
- The agent pushes a renewal into the clusters within minutes, or the External Secrets Operator does (serverless agent, on the roadmap)
- EKS Manager alerts if the renewal job stops succeeding

RUNNING IT · IN THE CLUSTERS

Every namespace gets the same certificate

Deploy places it; the agent keeps it current

Secrets Manager
one secret per zone

→

The agent
checks every ~2 minutes

→

dns-zone-certs
in each namespace

→

Ingress
names it under tls

Why the full chain

tls.crt holds the certificate plus its intermediate. Browsers forgive a missing intermediate; Go and Java clients, like the Argo CD CLI, do not.

Alerts if it falls behind

A namespace that stays on the old certificate raises an alert in EKS Manager, so a push that keeps failing is never silent.

On the roadmap: with the serverless agent, the External Secrets Operator in each cluster refreshes dns-zone-certs from Secrets Manager itself.

SECURITY

Narrow by design

Keys only the pipeline can read

EKSManagerLetsEncryptRole trusts codebuild.amazonaws.com and nothing else. No person or workflow can assume it.

TXT on _acme-challenge only

The example cert_manager policy writes TXT records named _acme-challenge.* in one public zone. It cannot repoint a hostname.

main is the boundary

Every workflow role requires refs/heads/main. Protect main with a required review on your private copy.

No silent deletes

Every certificate secret has prevent_destroy, so a zone dropped from the file can’t delete a live certificate.

The Terraform state holds the ACME account key and every private key. Its S3 bucket is as sensitive as the certificates.

ADDING A ZONE

Edit one file, run one workflow

No setup re-run. The order matters.

1 · Your role

Create cert_manager in the zone's account, trusting EKSManagerLetsEncryptRole.

2 · Push the file

Add the zone to hosted-zones.json. The sync grants the pipeline that role.

3 · Run lets-encrypt.yml

After the sync finishes. Run first and it fails on AssumeRole.

eksmanager.io | github.com/gitopsmanager/eksmanager-bootstrap